package com.twitter.search.common.util.earlybird;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

import com.google.common.base.Preconditions;

import com.twitter.search.adaptive.adaptive\_results.thriftjava.TweetSource;

import com.twitter.search.common.logging.ObjectKey;

import com.twitter.search.common.runtime.DebugManager;

import com.twitter.search.earlybird.thrift.EarlybirdRequest;

import com.twitter.search.earlybird.thrift.EarlybirdResponse;

import com.twitter.search.earlybird.thrift.EarlybirdResponseCode;

import com.twitter.search.earlybird.thrift.ThriftSearchQuery;

import com.twitter.search.earlybird.thrift.ThriftSearchResult;

import com.twitter.search.earlybird.thrift.ThriftSearchResults;

import com.twitter.search.earlybird.thrift.ThriftTweetSource;

/\*\* Utility methods that work on EarlybirdResponses. \*/

public final class EarlybirdResponseUtil {

private EarlybirdResponseUtil() {

}

/\*\*

\* Returns the results in the given EarlybirdResponse.

\*

\* @param response The EarlybirdResponse.

\* @return The results in the given EarlybirdResponse, or {@code null} if the response is

\* {@code null} or the results are not set.

\*/

public static ThriftSearchResults getResults(EarlybirdResponse response) {

if ((response == null) || !response.isSetSearchResults()) {

return null;

}

return response.getSearchResults();

}

/\*\*

\* Determines if the given EarlybirdResponse has results.

\*

\* @param response The EarlybirdResponse.

\* @return {@code true} if the given EarlybirdResponse has results; {@code false} otherwise.

\*/

public static boolean hasResults(EarlybirdResponse response) {

ThriftSearchResults results = getResults(response);

return (results != null) && results.isSetResults() && !results.getResults().isEmpty();

}

/\*\*

\* Returns the number of results in the given EarlybirdResponse.

\*

\* @param response The EarlybirdResponse.

\* @return The number of results in the given EarlybirdResponse.

\*/

public static int getNumResults(EarlybirdResponse response) {

return hasResults(response) ? response.getSearchResults().getResultsSize() : 0;

}

/\*\*

\* Determines the response is early-terminated.

\*

\* @param response The EarlybirdResponse.

\* @return {@code true} if the response is early-terminated; {@code false} otherwise.

\*/

public static boolean isEarlyTerminated(EarlybirdResponse response) {

Preconditions.checkNotNull(response);

return response.isSetEarlyTerminationInfo()

&& response.getEarlyTerminationInfo().isEarlyTerminated();

}

/\*\*

\* Returns if the response should be considered failed for purposes of stats and logging.

\*/

public static boolean responseConsideredFailed(EarlybirdResponseCode code) {

return code != EarlybirdResponseCode.SUCCESS

&& code != EarlybirdResponseCode.REQUEST\_BLOCKED\_ERROR

&& code != EarlybirdResponseCode.TIER\_SKIPPED;

}

/\*\*

\* Extract results from Earlybird response.

\*/

public static List<ThriftSearchResult> extractResultsFromEarlybirdResponse(

EarlybirdResponse response) {

return hasResults(response)

? response.getSearchResults().getResults() : Collections.emptyList();

}

/\*\*

\* Log the Earlybird response as a candidate source.

\*/

public static EarlybirdResponse debugLogAsCandidateSource(

EarlybirdResponse response, TweetSource tweetSource) {

List<ThriftSearchResult> results = extractResultsFromEarlybirdResponse(response);

debugLogAsCandidateSourceHelper(results, tweetSource);

return response;

}

/\*\*

\* Log a list of ThriftSearchResult as a candidate source.

\*/

public static List<ThriftSearchResult> debugLogAsCandidateSource(

List<ThriftSearchResult> results, TweetSource tweetSource) {

debugLogAsCandidateSourceHelper(results, tweetSource);

return results;

}

private static void debugLogAsCandidateSourceHelper(

List<ThriftSearchResult> results, TweetSource tweetSource) {

// debug message for Earlybird relevance candidate source

List<String> strIds = results

.stream()

.map(ThriftSearchResult::getId)

.map(Object::toString)

.collect(Collectors.toList());

ObjectKey debugMsgKey = ObjectKey.createTweetCandidateSourceKey(

tweetSource.name());

DebugManager.perObjectBasic(

debugMsgKey,

String.format("[%s][%s] results: %s", debugMsgKey.getType(), debugMsgKey.getId(), strIds));

}

/\*\*

\* Extract the real time response from an existing response

\*/

public static EarlybirdResponse extractRealtimeResponse(EarlybirdResponse response) {

EarlybirdResponse realtimeResponse = response.deepCopy();

if (EarlybirdResponseUtil.hasResults(response)) {

List<ThriftSearchResult> realtimeResults = realtimeResponse.getSearchResults().getResults();

realtimeResults.clear();

for (ThriftSearchResult result : response.getSearchResults().getResults()) {

if (result.getTweetSource() == ThriftTweetSource.REALTIME\_CLUSTER) {

realtimeResults.add(result);

}

}

}

return realtimeResponse;

}

/\*\*

\* Returns an EarlybirdResponse that should be returned by roots when a tier was skipped.

\*

\* @param minId The minSearchedStatusID to be set on the response.

\* @param maxId The maxSearchedStatusID to be set on the response.

\* @param debugMsg The debug message to be set on the response.

\* @return A response that should be returned by roots when a tier was skipped.

\*/

public static EarlybirdResponse tierSkippedRootResponse(long minId, long maxId, String debugMsg) {

return new EarlybirdResponse(EarlybirdResponseCode.SUCCESS, 0)

.setSearchResults(new ThriftSearchResults()

.setResults(new ArrayList<>())

.setMinSearchedStatusID(minId)

.setMaxSearchedStatusID(maxId))

.setDebugString(debugMsg);

}

/\*\*

\* Determines if the given response is a success response.

\*

\* A response is considered successful if it's not null and has either a SUCCESS, TIER\_SKIPPED or

\* REQUEST\_BLOCKED\_ERROR response code.

\*

\* @param response The response to check.

\* @return Whether the given response is successful or not.

\*/

public static boolean isSuccessfulResponse(EarlybirdResponse response) {

return response != null

&& (response.getResponseCode() == EarlybirdResponseCode.SUCCESS

|| response.getResponseCode() == EarlybirdResponseCode.TIER\_SKIPPED

|| response.getResponseCode() == EarlybirdResponseCode.REQUEST\_BLOCKED\_ERROR);

}

/\*\*

\* Finds all unexpected nullcast statuses within the given result. A nullcast status is

\* unexpected iff:

\* 1. the tweet is a nullcast tweet.

\* 2. the tweet is NOT explicitly requested with {@link ThriftSearchQuery#searchStatusIds}

\*/

public static Set<Long> findUnexpectedNullcastStatusIds(

ThriftSearchResults thriftSearchResults, EarlybirdRequest request) {

Set<Long> statusIds = new HashSet<>();

for (ThriftSearchResult result : thriftSearchResults.getResults()) {

if (resultIsNullcast(result) && !isSearchStatusId(request, result.getId())) {

statusIds.add(result.getId());

}

}

return statusIds;

}

private static boolean isSearchStatusId(EarlybirdRequest request, long id) {

return request.getSearchQuery().isSetSearchStatusIds()

&& request.getSearchQuery().getSearchStatusIds().contains(id);

}

private static boolean resultIsNullcast(ThriftSearchResult result) {

return result.isSetMetadata() && result.getMetadata().isIsNullcast();

}

}